17. Matching Cards Suppose two shuffled decks of cards are placed on a table, and then cards are drawn from the tops of the decks one at a time and compared. On average, how many matches do you think will occur? Write a program to carry out this process 10,000 times and calculate the average number of matches that occur. See Fig. 6.4. (***Note:*** This problem was first analyzed in 1708 by the French probabilist Pierre Remond de Montmort who determined that the theoretical answer is 1. There are many variations on the problem, and the theoretical answer is always 1 even when the number of items is other than 52. One variation of the problem is as follows: A typist types letters and envelopes to 20 different people. The letters are randomly put into the envelopes. On average, how many letters are put into the correct envelope?)

```
The average number of cards that
matched was 1.005659.
```

FIGURE 6.4 Possible outcome of Exercise 17.

```python
import random
import pickle

NUMBER_OF_TRIALS = 10000

def main():
    ## Carry out matching process NUMBER_OF_TRIALS times.
    totalNumberOfMatches = 0
    for i in range(NUMBER_OF_TRIALS):
        totalNumberOfMatches += matchTwoDecks()
    averageNumberOfMatches = totalNumberOfMatches / NUMBER_OF_TRIALS
    print("The average number of cards that")
    print("matched was {0:.3f}.".format(averageNumberOfMatches))

def matchTwoDecks():
    ## Determine the number of matches when comparing
    ## two shuffled decks of cards.
    # Create two decks as lists using the binary file
    # DeckOfCardsList.dat from Example 2.
    infile = open("DeckOfCardsList.dat", 'rb')
    deck1 = pickle.load(infile)
    infile.close()
    infile = open("DeckOfCardsList.dat", 'rb')
    deck2 = pickle.load(infile)
    infile.close()
    # Shuffle both decks of cards.
    random.shuffle(deck1)
    random.shuffle(deck2)
    # Compare cards and determine the number of matches.
    numberOfMatches = 0
    for i in range(52):
        if (deck1[i] == deck2[i]):
            numberOfMatches += 1
    return numberOfMatches

main()
```

Exercises 19 and 20 refer to the Powerball lottery. In the Powerball lottery, five balls are randomly selected from a set of white balls numbered 1 through 59, and then a single ball, called the Powerball, is randomly selected from a set of red balls numbered 1 through 35.

19. Powerball Lottery Write a program to randomly produce a Powerball drawing. See Fig. 6.6.

```
White balls: 22 28 51 11 5
Powerball: 20
```

FIGURE 6.6 Possible outcome of Exercise 19.

```
import random
## Simulate a Powerball Drawing.
whiteBalls = [num for num in range(1, 60)]
# Randomly sample and display five white balls.
whiteBallSelection = random.sample(whiteBalls, 5)
for i in range(5):
    whiteBallSelection[i] = str(whiteBallSelection[i])
print("White Balls:", " ".join(whiteBallSelection))
# Randomly select and display the Powerball.
powerBall = random.randint(1, 35)
print("Powerball:", powerBall)
```

20. Powerball Lottery Often the five selected white balls contain two or more balls with consecutive numbers. Write a program that simulates 100,000 selections of white balls and displays the percentage of times the selection contains at least two consecutive numbers. See Fig. 6.7. Feel free to use the function `random.sample((range(1,60), 5))`.

```
31% of the time there were at least
two consecutive numbers in the set
of five numbers.
```

FIGURE 6.7 Possible outcome of Exercise 20.

```
import random

def main():
    total = 0
    for trial in range(100000):
        L = [n for n in range(1, 60)]
        numbers = random.sample(L, 5)
        # Can replace above two lines with
        # numbers = random.sample(range(1,60), 5)
        numbers.sort()
        if two_consecutive(numbers):
            total += 1
    sentence = " of the time there were at least \ntwo consecutive numbers" + \
               "in the set \nof five numbers."
    print(("{0:.0%}" + sentence).format(total / 100000))

def two_consecutive(x):
    for index in range(0, 4):
        if x[index] + 1 == x[index + 1]:
            return True

main()
```

21. Coin Toss Write a program to display the result of tossing a coin 32 times. Then determine if there is a run of five consecutive Heads or a run of five consecutive Tails. See Fig. 6.8. (***Note***: When you toss a coin 2^r times, you are more likely than not to have a run of r Heads or r Tails.)

```
THTHHTHHTTTTHTHHHHTTHHHHHTTHHHTH
There was a run of five consecutive
same outcomes.
```

FIGURE 6.8 Possible outcome of Exercise 21.

```
The average number of cards
turned up was 10.61.
```

FIGURE 6.9 Possible outcome of Exercise 22.

```
import random
## Simulate 32 coin tosses and check for runs of length five.
coin = ['T', 'H']
result = ""
for i in range(32):
    result += random.choice(coin)
print(result)
if ("TTTTT" in result) or ("HHHHH" in result):
    print("There was a run of five consecutive")
    print("same outcomes.")
else:
    print("There was no run of five consecutive same outcomes.")
```

22. Locate First Ace Suppose you shuffle an ordinary deck of 52 playing cards, and then turn up cards from the top until the first ace appears. On average, how many cards do you think must be turned up until the first ace appears? Estimate the average by writing a program that shuffles a deck of cards 100,000 times and finds the average of the number of cards that must be turned up to obtain an ace for each shuffle. See Fig. 6.9. ***Note:*** Your average will differ from the one in the figure, but should be close to 10.6.

```
import random

## Think of the cards as being numbered 1 through 52.
cards = [n for n in range(1, 53)]
total = 0
for i in range(100000):
    aceLocations = random.sample(cards, 4)
    n = min(aceLocations)
    total += n
print("The average number of cards \nturned up was {0:.2f}".
      format(total / 100000))
```

23. Bridge (HPC) A bridge hand consists of 13 cards. One way to evaluate a hand is to calculate the total high point count (HPC) where an ace is worth four points, a king is worth three points, a queen is worth two points, and a jack is worth one point. Write a program that randomly selects 13 cards from a deck of cards and calculates the HPC for the hand. See Fig. 6.10. ***Note:*** Use the pickled file `DeckOfCardsList.dat`.

```
7♥, A♦, Q♠, 4♣, 8♠, 8♥, K♠, 2♦, 10♦, 9♦, K♥, Q♦, Q♣
HPC = 16
```

FIGURE 6.10 Possible outcome of Exercise 23.

```
import random
import pickle

def main():
    ## Calculate the High Point Count for a bridge hand.
    bridgeHand = getHand()
    print(", ".join(bridgeHand))  # Display the bridge hand.
    HCP = calculateHighCardPointCount(bridgeHand)
    print("HPC =", HCP)

def getHand():
    infile = open("DeckOfCardsList.dat", 'rb')
    deckOfCards = pickle.load(infile)
    infile.close()
    bridgeHand = random.sample(deckOfCards, 13)
    return bridgeHand

def calculateHighCardPointCount(bridgeHand):
    countDict = {'A':4, 'K':3, 'Q':2, 'J':1}
    HPC = 0
    for card in bridgeHand:
        rank = card[0]  # Each card is a string of
                        # two characters.
    if rank in "AKQJ":
        HPC += countDict[rank]
    return HPC

main()
```

In Exercise 24, use the file `StatesANC.txt` that contains the name, abbreviation, nickname, and capital of each state in the United States. The states are listed in alphabetical order. The first three lines of the file are

```
Alabama,AL,Cotton State,Montgomery
Alaska,AK,The Last Frontier,Juneau
Arizona,AZ,Grand Canyon State,Phoenix
```

24. State Capital Quiz Write a program that asks the user to name the capitals of five randomly chosen states. The program should then report the number of incorrect answers and display the answers to the missed questions. See Fig. 6.11.

```
What is the capital of Minnesota? Saint Paul
What is the capital of California? Sacramento
What is the capital of Illinois? Chicago
What is the capital of Alabama? Montgomery
What is the capital of Massachusetts? Boston

You missed 1 question.
Springfield is the capital of Illinois.
```

FIGURE 6.11 Possible outcome of Exercise 24.

```python
import random

numbers_used = []     # will prevent a number from being repeated
numbers_missed = []   # numbers of questions missed
infile = open("StatesANC.txt", 'r')
state_data = [line.rstrip() for line in infile]
infile.close()
# Get five different randomly selected numbers.
for num in range(5):
    num = random.randint(0, 49)
    while num in numbers_used:
        num = random.randint(0, 49)
    numbers_used.append(num)
    item = state_data[num].split(',')
    capital = input("What is the capital of " + item[0] + "? ")
    if capital != item[3]:
        numbers_missed.append(num)
# give score and corrections
print()
total_number_missed = len(numbers_missed)
if total_number_missed == 0:
    print("Congratulations. You answered every question correctly.")
else:
    if total_number_missed == 1:
        print("You missed 1 question.")
    else:
        print("You missed", total_number_missed, "questions.")
    for number in numbers_missed:
        item = state_data[number].split(',')
        print(item[3], "is the capital of", item[0])
```

1. Guess My Number Write a robust program that randomly selects a number from 1 through 100 and asks the user to guess the number. At each guess the user should be told if the guess is proper, and if so, whether it is too high or too low. The user should be told of the number of guesses when finally guessing the correct number. See Fig. 6.28.

```
I've thought of a number from 1 through 100.
Guess the number: 50
Too low
Try again: 123
Number must be from 1 through 100.
Try again: sixty
You did not enter a number.
Try again: 60
Too high
Try again: 56
Correct. You took 5 guesses.
```

```python
import random

numberOfTries = 1
n = random.randint(1, 100)
print("\nI've thought of a number from 1 through 100.")
while True:
    try:
        guess = int(input("Guess the number: "))
        break
    except ValueError:
        numberOfTries += 1
        print("You did not enter a number.")
while guess != n:
    numberOfTries += 1
    if (guess > 100) or (guess < 1):
        print("Number must be from 1 through 100.")
    elif guess > n:
        print("Too high")
    elif guess < n:
        print("Too low")
    while True:
        try:
            guess = int(input("Try again: "))
            break
        except ValueError:
            numberOfTries += 1
            print("You did not enter a number.")
print("Correct.", end=" ")
if numberOfTries == 1:
    print("You got it in one guess.")
else:
    print("You took", numberOfTries, "guesses.")
```

2. Analyze a Poker Hand Write a program using the file `DeckOfCardsList.dat` that randomly selects and displays five cards from the deck of cards and determines which of the following seven categories describes the hand: four-of-a-kind, full house (three cards of one rank, two cards of another rank), three-of-a-kind, two pairs, one pair, or ranks-all-different. See Fig. 6.29. (*Hint:* Determine the number of different ranks in the hand and analyze each of the four possible cases.)

```
K♥, K♦, 2♦, K♣, 5♠
three-of-a-kind
```

FIGURE 6.29 Possible outcome of Programming Project 2.

```python
import pickle
import random

def main():
    pokerHand = getHandOfCards(5)
    pokerHand.sort()
    displayPokerHand(pokerHand)
    analyzePokerHand(pokerHand)

def getHandOfCards(numberOfCards):
    deckOfCards = pickle.load(open("deckOfCardsList.dat", 'rb'))
    return random.sample(deckOfCards, 5)

def displayPokerHand(pokerHand):
    print(", ".join(pokerHand))

def analyzePokerHand(pH):
    ranks = {x[:-1] for x in pH}
    numberOfRanks = len(ranks)
    if numberOfRanks == 5:
        print("ranks-all-different")
    elif numberOfRanks == 4:
        print("one pair")
    elif numberOfRanks == 3:
        foundThree = False
        for i in range(2):
            if ((pH[i][0] == pH[i + 1][0]) and
                (pH[i + 1][0] == pH[i + 2][0])):
                print("three of a kind")
                foundThree = True
                break
        if foundThree == False:
            print("two pairs")
    else:  # two different ranks
        if ((pH[0][0] == pH[1][0]) and (pH[1][0] == pH[2][0]) \
             and (pH[2][0] == pH[3][0])) \
             or ((pH[1][0] == pH[2][0]) and (pH[2][0] == pH[3][0]) \
             and (pH[3][0] == pH[4][0])):
            print("four of a kind")
        else:
            print("full house")

main()
```

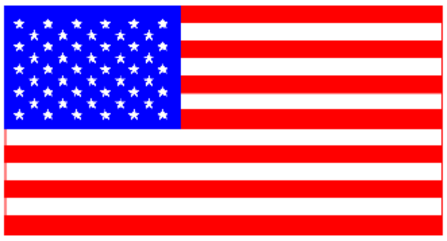
3. **Analyze a Bridge Hand** Write a program using the file `DeckOfCardsList.dat` that randomly selects and displays 13 cards from the deck of cards and gives the suit distribution. See Fig. 6.30.

```
10♥, 3♥, J♣, 2♣, 10♦, K♣, 2♥, 6♦, 6♣, 4♣, 7♦, 6♠, 4♦
Number of ♣ is 5
Number of ♦ is 4
Number of ♥ is 3
Number of ♠ is 1
```

FIGURE 6.30 Possible outcome of Programming Project 3.

```python
import pickle
import random

def main():
    ## Analyze a bridge hand.
    bridgeHand = getHandOfCards(13)
    displayBridgeHand(bridgeHand)
    analyzeBridgeHand(bridgeHand)

def getHandOfCards(numberOfCards):
    deckOfCards = pickle.load(open("deckOfCardsList.dat", 'rb'))
    return random.sample(deckOfCards, numberOfCards)

def displayBridgeHand(bridgeHand):
    print(", ".join(bridgeHand))

def analyzeBridgeHand(bridgeHand):
   suits = {x[-1] for x in bridgeHand}
   d = {suit:0 for suit in suits}  # distribution of cards into suits
   for card in bridgeHand:
       d[card[-1]] += 1
   t = tuple(d.items())
   tSorted = sorted(t)
   tSorted = sorted(t, key=lambda x: x[1], reverse=True)
   for k in tSorted:
       print("Number of", k[0], "is", k[1])

main()
```

4. American Flag The width (w) of the official American flag is 1.9 times the height (h). The blue rectangular canton (referred to as the "union") has width $\frac{2}{5}w$ and height $\frac{7}{13}h$. Write a program that draws an American flag. See Fig. 6.31. The colorful insert pages contain a picture of the flag with its true colors.



```
import turtle

def main():
    ## Draw an American flag.
    t = turtle.Turtle()
    t.hideturtle()
    t.speed(10)
    drawRectangle(t, -200, 0, 380, 200, "red", "red")
    for i in range(1, 13, 2):
        drawRectangle(t, -200, (200/13)*i , 380, (200/13), "red", "white")
    drawRectangle(t, -200, (200/13)*6, (2/5)*380, (200/13)*7, "blue", "blue")
    for i in range(5):
        y = 180 - (20 * i)
        for j in range(6):
            x = -190 + 25*j
            drawFivePointStar(t, x, y, 8, "white")
    for i in range(4):
        y = 170 - (20 * i)
        for j in range(5):
            x = -177 + 25*j
            drawFivePointStar(t, x, y, 8, "white")

def drawFivePointStar(t, x, y, lenthOfSide, colorP="black", colorF="white"):
    t.up()
    t.goto(x, y)
    t.setheading(0)
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.left(36)
    t.down()
    t.begin_fill()
    for i in range(6):
        t.forward(lenthOfSide)
        t.left(144)
    t.end_fill()
```

```
def drawRectangle(t, x, y, w, h, colorP="black", colorF="white"):
    # Draw a rectangle with bottom-left corner (x, y),
    # width w, height h, pencolor colorP, and fill color colorF.
    originalPenColor = t.pencolor()
    originalFillColor = t.fillcolor()
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x , y)          # bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)       # bottom-right corner of rectangle
    t.goto(x + w, y + h)   # top-right corner of rectangle
    t.goto(x, y + h)       # top-left corner of rectangle
    t.goto(x , y)          # bottom-left corner of rectangle
    t.end_fill()
    t.pencolor(originalPenColor)
    t.fillcolor(originalFillColor)

main()
```

5. Permutations A reordering of the letters of a word is called a *permutation* of the word. A word of n different characters has $n!$ permutations where $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$. For instance, the word *python* has 6! or 720 permutations. Some of its permutations are *pythno*, *ypntoh*, *tonyhp*, and *ontphy*. Write a program that requests a word without repeated characters as input and then displays all the permutations of the word. See Fig. 6.32. (***Hint:*** Suppose the word has six characters. Consider the characters of the word one at a time. Then display the words beginning with that character and followed by each of the 5! permutations of the remaining characters of the word.)

```
Enter a word: ear
ear   era   aer   are   rea   rae
```

```
def main():
    ## Determine all the permutations of a word.
    w = input("Enter a word: ")
    print("   ".join(permutations(w)))

def permutations(w):
    # Construct a list consisting of all permutations of the string s
    if len(w) == 1:
        return w
    listOfPermuations = []          # resulting list
    for i in range(len(w)):
        restOfw = w[:i] + w[i+1:]   # w with ith character removed
        z = permutations(restOfw)   # permutations of remaining characters
        for t in z:
            listOfPermuations.append(w[i] + t)
    return listOfPermuations

main()
```

6. Pascal's Triangle The triangular array of numbers in Fig. 6.33 is called *Pascal's triangle*, in honor of the seventeenth century mathematician Blaise Pascal. The n^{th} row of the triangle gives the coefficients of the terms in the expansion of $(1 + x)^n$. For instance, the 5^{th} row tells us that

$$(1 + x)^5 = 1 + 5x + 10x^2 + 10x^3 + 5x^4 + 1x^5$$

```
                                      Row
               1                       0
            1     1                    1
         1     2     1                 2
      1     3     3     1              3
   1     4     6     4     1           4
1     5    10    10     5     1        5
```

FIGURE 6.33 Pascal's Triangle.

With the coefficients arranged in this way, each number in the triangle is the sum of the two numbers directly above it (one to the left and one to the right). For example, in row four, 1 is the sum of 1 (the only number above it), 4 is the sum of 1 and 3, 6 is the sum of 3 and 3, and so on. Since each row can be calculated from the previous row, recursion can easily be used to generate any row of Pascal's triangle. Write a program that prompts the user for a nonnegative integer *n* and then displays the numbers in the n^{th} row of the triangle. See Fig. 6.34.

```
Enter a nonnegative integer: 6
Row 6: 1 6 15 20 15 6 1
```

FIGURE 6.34 Possible outcome of Programming Project 6.

```
def main():
    ## Display line of Pascal's Triangle.
    n = int(input("Enter a nonnegative integer: "))
    line = [str(x) for x in pascal(n)]
    print("Row", str(n) + ':' ," ".join(line))

def pascal(n):
    # Display the nth line of Pascal's triangle.
    if n == 0:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])
        line += [1]
    return line

main()
```